Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студенты группы 22ВВВ2:

Зубриянова А.А.

Кондратьева В.И.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Оценка времени выполнения программ.

**Цель работы**

Изучение.

**Лабораторное задание**

Задание 1:

1. Вычислить порядок сложности программы (*О*-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

Задание 2:

Сортировка Шелла и быстрая сортировка.

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая – убывающую.
5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Описание метода решения задачи**

Задание 1:

1. Вычислили порядок сложности программы.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A = O(1)  B = O(n)  C = O(1)  D = O(n)  E = O(n2)  F = O(n3) | G = O(1)  H = O(n)  I = O(1)  J = O(n)  K = O(n2) | L = O(1) | M = O(1)  N = O(n)  O = O(1)  P = O(n)  Q = O(n2) | R = O(1) |
| S = O(n3) | | | | |

Порядок сложности программы: O(n3)

1. Используя команды

clock\_t start = clock();

clock\_t end = clock();

double time = (double)(end - start)/ CLOCKS\_PER\_SEC;

оценили время выполнения программы и кода, выполняющего перемножение матриц.

1. Таблица зависимости времени выполнения программы от размера матриц.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | 100 | 200 | 400 | 1000 | 2000 | 4000 | 10000 |
| время | 0,004 | 0,024 | 0,284 | 4,021 | 47,722 | 869,145 | 5693,753 |
| O(n^3) | 0,009 | 0,072 | 0,576 | 9 | 72 | 576 | 9000 |

Const = 0,000000009

Задание 2:

1. Используя команды

clock\_t start = clock();

clock\_t end = clock();

double time = (double)(end - start)/ CLOCKS\_PER\_SEC;

оценили время работы алгоритма сортировки Шелла, быстрой сортировки и стандартной функции qsort на случайном наборе значений массива; на массиве, представляющем собой возрастающую последовательность чисел; на массиве, представляющем собой убывающую последовательность чисел; на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая – убывающую.

**Таблицы сравнения времени работы сортировок (50000; 100000; 20000).**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 0,347 | 0,001 | 0,691 | 0,346 |
| Быстрая сортировка | 0,006 | 0,003 | 0,003 | 0,125 |
| qsort | 0,010 | 0,008 | 0,008 | 0,028 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 1,375 | 0,003 | 2,732 | 1,361 |
| Быстрая сортировка | 0,012 | 0,006 | 0,006 | 0,498 |
| qsort | 0,019 | 0,015 | 0,017 | 0,062 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 5,527 | 0,004 | 8,424 | 3,865 |
| Быстрая сортировка | 0,024 | 0,013 | 0,010 | 0,859 |
| qsort | 0,033 | 0,033 | 0,024 | 0,097 |

Исходя из данных таблицы, можно сделать вывод о том, что для сортировки массива случайных чисел лучше выбирать быструю сортировку, для возрастающей последовательности – сортировку Шелла, для убывающей последовательности – быструю сортировку, а для смешанной последовательности – сортировку qsort.

**Листинг**

Задание 1:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

int main(void)

{

setlocale(LC\_ALL, "Russian");

int N;

printf("Введите размер массива: ");

scanf("%d", &N);

int\*\* a = (int\*\*)malloc(sizeof(int\*) \* N);

for (int i = 0; i < N; i++) a[i] = (int\*)malloc(sizeof(int) \* N);

int\*\* b = (int\*\*)malloc(sizeof(int\*) \* N);

for (int i = 0; i < N; i++) b[i] = (int\*)malloc(sizeof(int) \* N);

int\*\* c = (int\*\*)malloc(sizeof(int\*) \* N);

for (int i = 0; i < N; i++) c[i] = (int\*)malloc(sizeof(int) \* N);

int i = 0, j = 0, r, elem\_c;

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while (i < N)

{

while (j < N)

{

a[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i = 0; j = 0;

while (i < N)

{

while (j < N)

{

b[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

clock\_t start = clock();

for (i = 0;i < N;i++)

{

for (j = 0;j < N;j++)

{

elem\_c = 0;

for (r = 0;r < N;r++)

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

c[i][j] = elem\_c;

}

}

//printf("%d%% ", i\*100/N);

}

clock\_t end = clock();

double time = (double)(end - start)/ CLOCKS\_PER\_SEC;

printf("Время выполнения: %lf\n", time);

for (int i = 0; i < N; ++i) free(a[i]);

free(a);

for (int i = 0; i < N; ++i) free(b[i]);

free(b);

for (int i = 0; i < N; ++i) free(c[i]);

free(c);

return(0);

}

Задание 2:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <windows.h>

#include <conio.h>

void calc(int\* a, int n);

void shell(int\* items, int count);

void output(int\* items, int count);

void qs(int\* items, int left, int right);

int compare(const void\* x1, const void\* x2);

int main()

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int N;

printf("Введите размер массива: ");

scanf("%d", &N);

int\* A = (int\*)malloc(sizeof(int) \* N);

//Случайный набор значений массива А

for (int i = 0; i < N; i++)

{

A[i] = rand() % 200 - 100;

//printf("%d ", A[i]);

//printf("\n");

}

printf("\n");

printf("Для массива из случайных чисел\n");

printf("\n");

calc(A, N);

printf("\n");

printf("\n");

//По возрастанию

for (int i = 0; i < N; i++)

{

A[i] = i\*2 - N;

}

//output(A, N);

printf("Для возрастающей последовательности\n");

printf("\n");

calc(A, N);

printf("\n");

printf("\n");

//По убыванию

for (int i = 0; i < N; i++)

{

A[i] = -(i \* 2 - N);

}

//output(A, N);

printf("Для убывающей последовательности\n");

printf("\n");

calc(A, N);

printf("\n");

printf("\n");

//Половина убывает, половина возрастает

for (int i = 0; i < N; i++)

{

A[i] = abs(i \* 2 - N);

}

//output(A, N);

printf("Для смешанной последовательности\n");

printf("\n");

calc(A, N);

free(A);

return 0;

}

void calc(int\* a, int n)

{

clock\_t start, end;

double time;

int\* B = (int\*)malloc(sizeof(int) \* n);

memcpy(B, a, sizeof(int) \* n);

start = clock();

shell(B, n);

end = clock();

time = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Сopтировка Шелла:%lf\n", time);

memcpy(B, a, sizeof(int) \* n);

start = clock();

qs(B, 0, n - 1);

end = clock();

time = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Быстрая сортировка:%lf\n", time);

memcpy(B, a, sizeof(int) \* n);

start = clock();

qsort(B, n, sizeof(int), compare);

end = clock();

time = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("qsort:%lf\n", time);

free(B);

}

void output(int\* items, int count)

{

for (int i = 0; i < count; i++)

{

printf("%d ", items[i]);

}

printf("\n");

}

void shell(int\* items, int count)

{

int i, j, gap, k;

int x, a[5];

a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

for (k = 0; k < 5; k++)

{

gap = a[k];

for (i = gap; i < count; ++i)

{

x = items[i];

for (j = i - gap; (x < items[j]) && (j >= 0); j = j - gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

void qs(int\* items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

i = left; j = right;

/\* выбор компаранда \*/

x = items[(left + right) / 2];

do {

while ((items[i] < x) && (i < right)) i++;

while ((x < items[j]) && (j > left)) j--;

if (i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while (i <= j);

if (left < j) qs(items, left, j);

if (i < right) qs(items, i, right);

}

int compare(const void\* x1, const void\* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

**Результаты работы программы**

Результаты работы программы представлены на рисунках:

Задание 1:

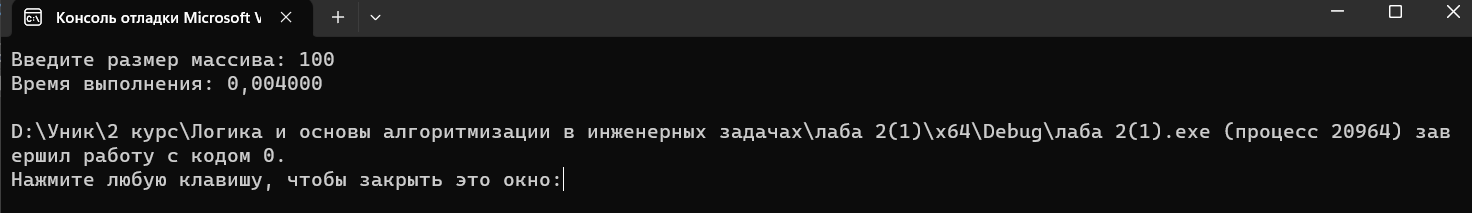


Рисунок 1 – для матрицы 100 на 100

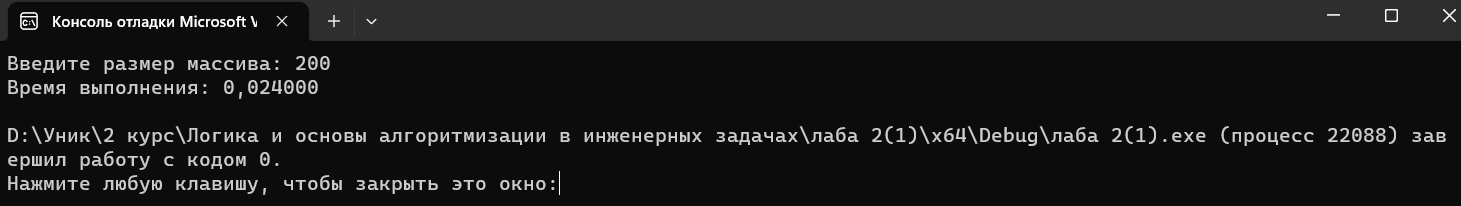


Рисунок 2 – для матрицы 200 на 200

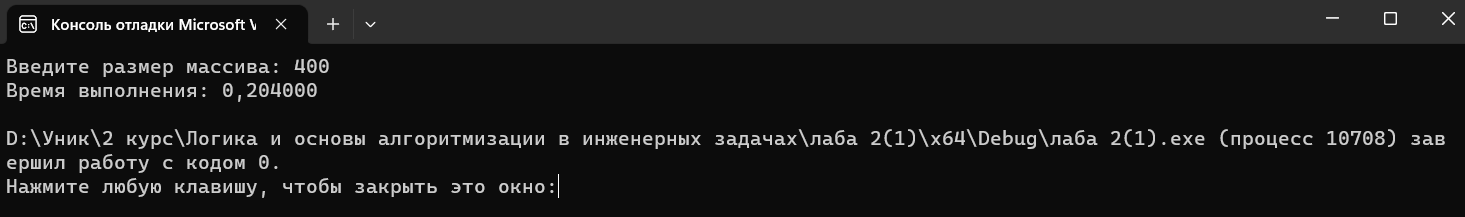


Рисунок 3 – для матрицы 400 на 400

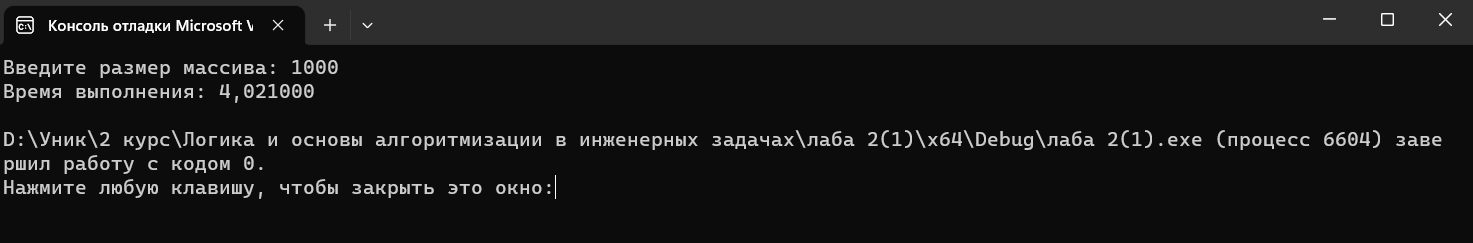


Рисунок 4 – для матрицы 1000 на 1000

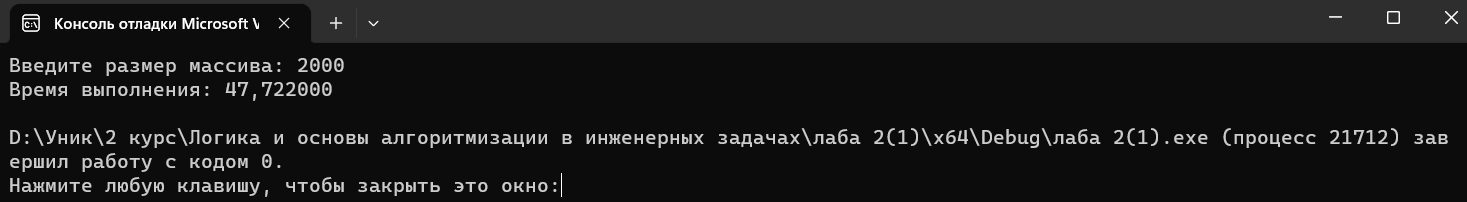


Рисунок 5 – для матрицы 2000 на 2000

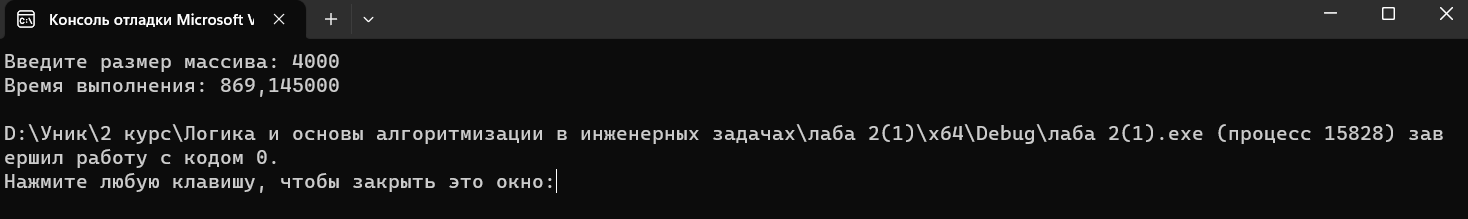


Рисунок 6 – для матрицы 4000 на 4000

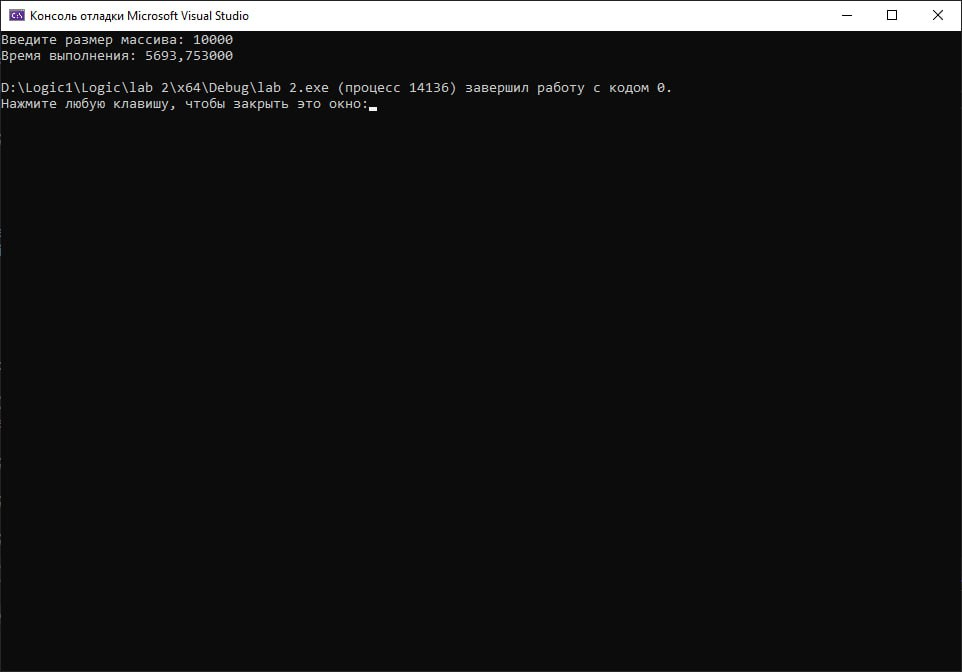


Рисунок 7 – для матрицы 10000 на 10000

Задание 2:

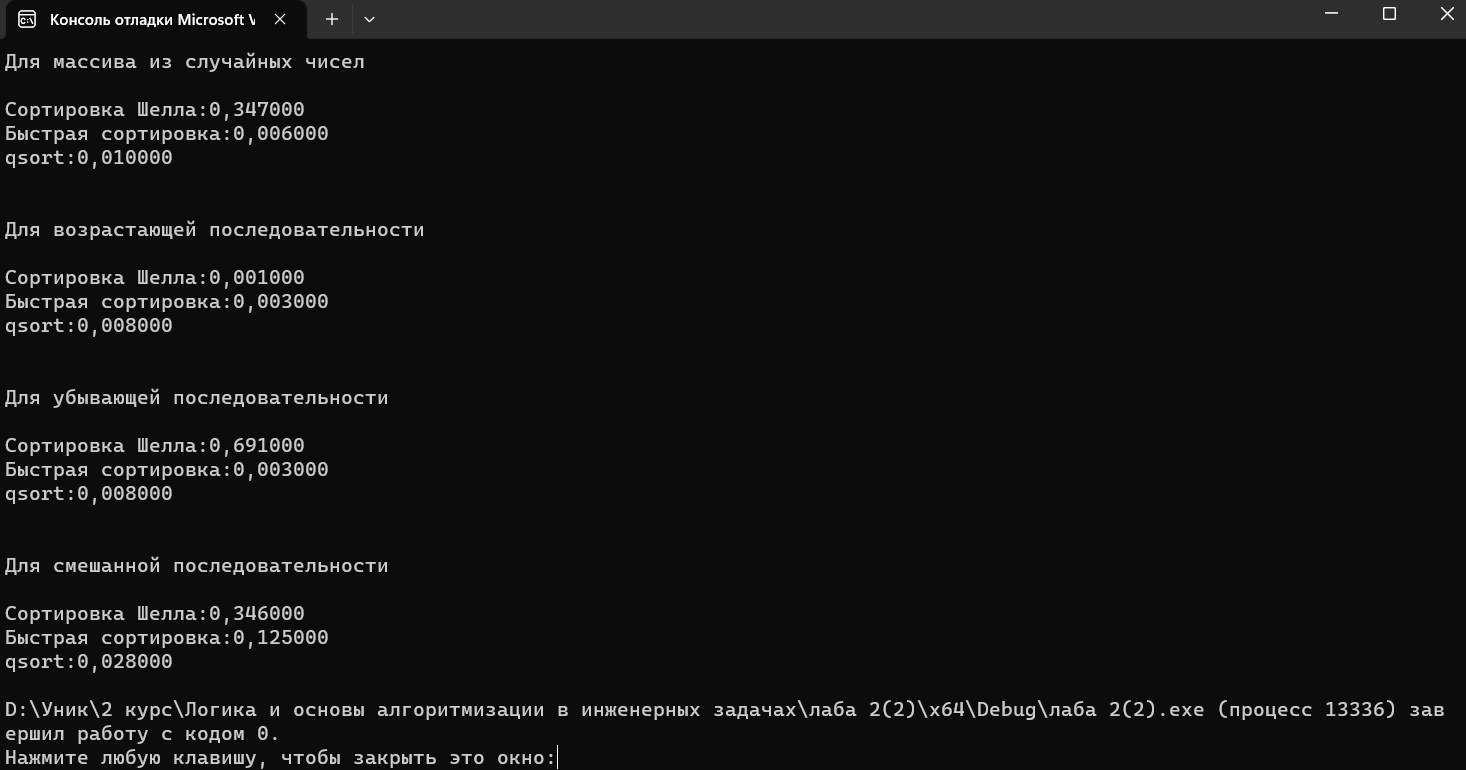


Рисунок 8 – для массива из 50000 элементов

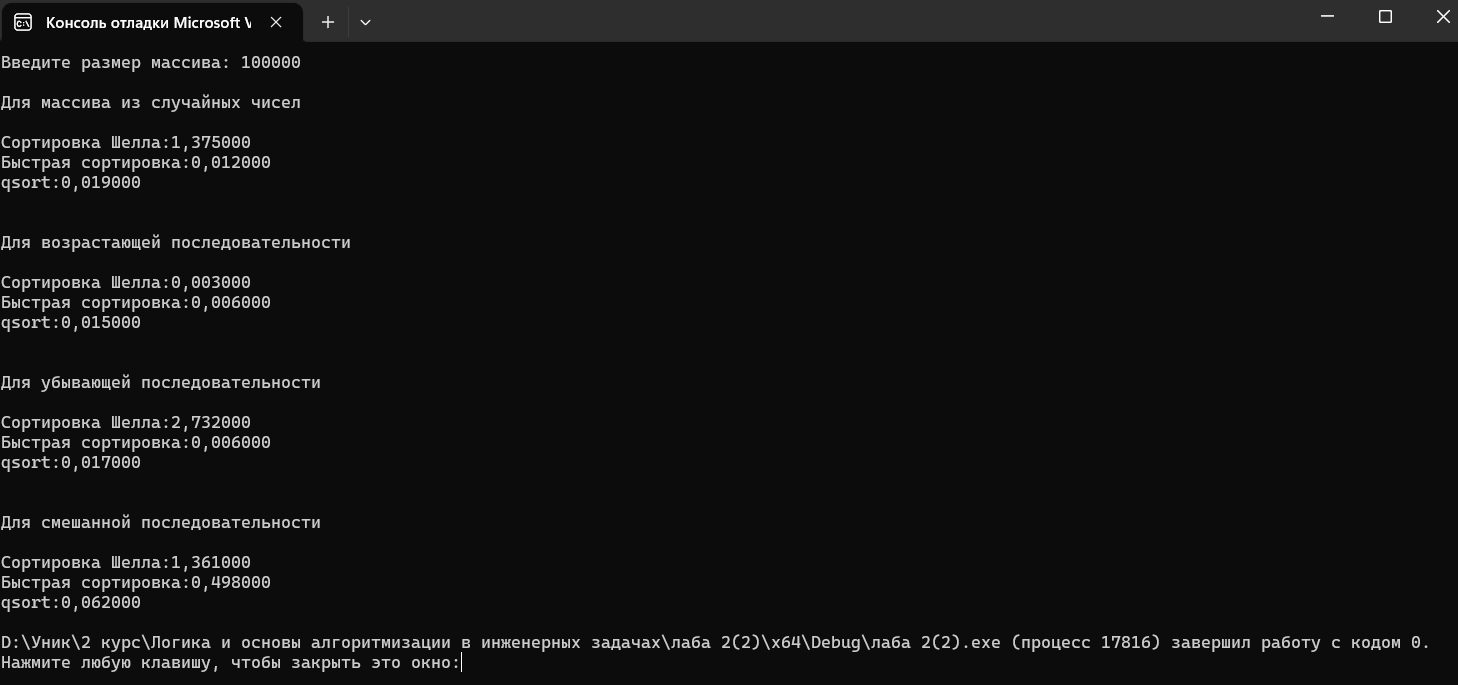


Рисунок 9 – для массива из 100000 элементов

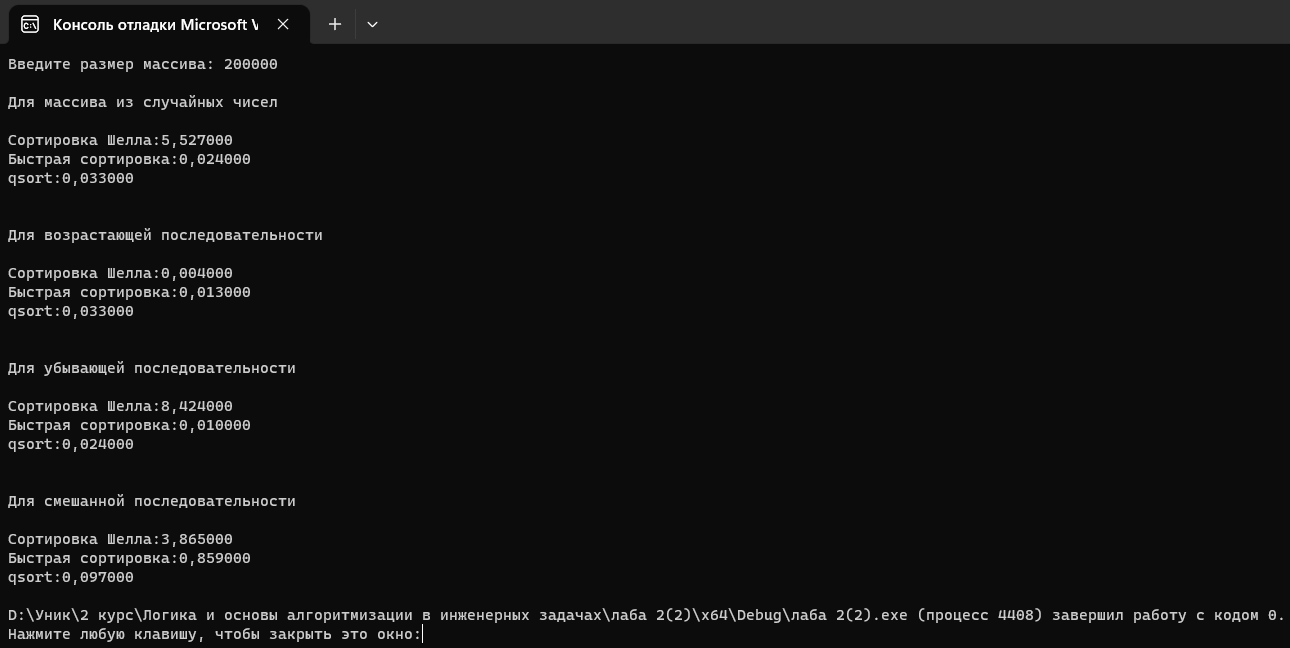


Рисунок 10 – для массива из 200000 элементов

**Вывод**

В ходе выполнения лабораторной работы были разработаны алгоритмы, оценивающие время выполнения работы программ. Результаты работы программы совпали с ожидаемыми результатами, следовательно, программа работает без ошибок.

Получили опыт в создании проектов в среде MicrosoftVisualStudio, приобрели навыки программирования алгоритмов.